End-to-End Real-Time Collision Avoidance Using Clustering Algorithms

Saym Imtiaz
simtiaz@wisc.edu

1. Introduction

Remote control of a robotic manipulator is useful for a variety of tasks such as telesurgery, manufacturing, and home environments. However, teleoperation in these diverse environments presents a multitude of unique challenges, primarily following human commands in real-time while avoiding various static and dynamic obstacles. There has been much research into both challenges. In terms of achieving motion, CollisionIK addresses achieving per-instant pose and motion objectives while avoiding kinematic singularities and maintaining continuity of joint configurations. Likewise, works such as OctoMap and AtomMap address modeling static and dynamic obstacles from raw sensor data by utilizing different forms of occupancy grids. The main limitation is that there are very few works that combine both problems. For example, CollisionIK can deal with static and dynamic obstacles in real-time but requires the end user to process raw sensor data and convert obstacles into basic shapes or meshes. As such, it contains too much data to be handled effectively by real-time motion solvers.

This project aims to bridge the gap between environment mapping, real-time motion planning, and collision avoidance by creating an end-to-end collision avoidance system. The initial proposal was to utilize a Kinect v2 and Kinova MOVO to accomplish this goal; however, this was switched to a UR5 and Intel RealSense D435 camera. This project first proposes to calibrate and scan the working environment with raw point cloud data from the D435 camera. Then, this point cloud data is processed and filtered to prepare for object extraction. This project proposes a novel multi-level clustering method to quickly decompose the point cloud into a collection of axis-aligned bounding boxes (AABBs) that can quickly and efficiently be processed by the motion solver. The real-time motion solving and collision avoidance is then handled by CollisionIK, thus bridging the gap between environment mapping, real-time motion planning, and static environmental collision avoidance. This project also sought to implement a method potentially extendable to dynamic obstacles as well.

2. Related Works

This section highlights prior works to which this system drew inspiration and guidance from in the areas of environment mapping, object extraction, and collision avoidance.

2.1. Occupancy Grids

An occupancy grid is a 3D space represented as a grid, where each voxel in the grid can be independently considered as occupied, free, or unexplored. Occupancy probabilities can be estimated from point clouds and are updatable with each new measurement. They are advantageous as they quantize space into a finite number of voxels rather than an infinite amount of infinitesimal space. They are often used in robotics and can be used for tasks such as mapping, localization, motion planning, and obstacle avoidance [8].

OctoMap is probabilistic 3D occupancy grid mapping approach that relies on an octree data structure [9]. OctoMap also includes a compression method to reduce the memory requirements. AtomMap extends OctoMap by relaxing the constraint that the occupancy grid must be uniform and tessellated [8]. Instead, AtomMap represents the occupancy grid with a collection of non-overlapping, equally-sized spheres. This approach more accurately represents the surface of objects by using an interpolation method across spheres. Occupancy grid updates suffer from slow updates in real-time applications. Kwon et al. [11] presents a method to reduce the number of points needed to update the occupancy grids by proposing a super ray and culling regions approach. However, these approaches do not present a clear avenue on how to implement with real-time collision avoidance techniques.

2.2. Bounding Volume Hierarchies

A common approach for collision detection is the use of bounding volume hierarchies (BVH). Dinas et al. [6] discusses how BVH can be used for broad-phase and narrow-phase collision detection methods. The work also discusses how a BVH should fit the underlying object as tightly as possible and how a BVH can encapsulate child nodes to speed up narrow-phase collision detection methods. The system developed utilizes a bounding volume method to

quickly model the environment and efficiently avoid collisions.

CAB [14] presents a BVH-based method to utilize Oriented Bounding Box (OBB) Trees to speed up collision detection between dynamic articulated bodies. However, unlike this work, this system utilizes axis-aligned bounding boxes (AABBs) and only attempts collision avoidance on static objects. This is due to the advantages present in using OBB trees are only relevant to dynamic bodies where rotational invariance is essential.

2.3. Collision Avoidance Methods

There has been much research in the fields of robotics and drone control in regards to collision avoidance and generation motion plans.

Some methods utilize an OctoMap in order to generate real-time motion. 3DVFH+ [15] assumes an OctoMap has already been created of the operating environment and utilizes a polar histogram to generate real-time collision avoidance for a flying robot. It therefore cannot handle dynamic obstacles. Fjeld [7] utilizes an OctoMap and used a learning based method to generate real-time collision avoidance, but was unsuccessful in getting their learning algorithm to work correctly. CollisionIK [13] is able to handle both static and dynamic obstacles, and can generate real-time motions with collision avoidance. However, it does not include an easy method to incorporate sensor data. RCIK [10] is also able to handle both static and dynamic obstacles, an also has the advantage of including a means of incorporating real-time sensor data by utilizing a much quicker occupancy grid. However, RCIK struggles with robots with higher degrees-of-freedom (DOF). This work builds upon the work of CollisionIK by implementing a compatible bounding volume based environment representation to enable an end-to-end collision avoidance system for static obstacles.

2.4. Clustering Algorithms

One technique to partition point cloud data is into clusters using clustering algorithms. One strategy is to use density-based clustering algorithms such as DBSCAN [1], which split point clouds into clusters based on the notion of noise. Another popular approach is K-means based algorithms [4] [3]. These work by having the user specify the number of clusters the data will be partitioned into, known as k. These algorithms can also be optimized by selecting the initial cluster centroids in a manner that avoids super-polynomial worst-case runtime, known as k-means++ [2]. This work utilizes both approaches in a novel multi-level clustering technique to take advantage of both types of techniques, as well as considers potentially optimal hyperparameter selection such as k-means++.

Figure 1. The D435 camera UR5 end-effector mount.

3. Technical Approach

This section provides an overview of the approach used by the system, the strengths and weaknesses of the approach, and some of the analysis done to develop and optimize the approach.

3.1. Calibration

The first aspect of the collision avoidance system is the calibration of the camera to the reference frame of the robot. This is important as it allows the object detection system to properly locate objects for the robot to avoid. For this project, eye-on-hand calibration was performed, meaning that a camera mounted on the end-effector of the UR5 was calibrated to the UR5 base frame. The camera mount is shown in Figure 1.

The calibration was done with a calibration package titled "ur5_realsense_calibration" from GitHub user "portgas-ray" [12]. This package is specifically designed for UR5 calibration using any of the Intel Realsense cameras. It works by first placing an ArUco marker on any flat surface in view of the D435 camera. It then requires the UR5 to move to various tracking positions while it records samples for the calibration calculation. It typically will require at least 20 different samples to be accurate, and it is important each sample taken has a clear view of the marker and each sample is verified to be accurate. After taking the samples, the package will compute the transform, and this transform can then be used in the "pclTransform" package to properly transform any point cloud scans taken by the D435. This package could also be used for eye-on-base calibration as well, in which the D435 could be mounted on a tripod and then calibrated to the UR5. The key limitation to this method is that the package is sensitive to the quality of samples taken. If any of the samples were not properly taken, the entirety of the calculation could be significantly inaccu-

rate.

3.2. Point Cloud Processing

The next aspect of the collision avoidance system is to acquire and transform the point cloud scan for object extraction. This is done by first defining poses for the UR5 to capture point cloud frames at in the scanMover.py file. Figure 2 shows the UR5 in the process of scanning the working environment using this script. Then, the UR5 is initialized and moved to each pose, and then capture a point cloud frame at each location. Each frame is then transformed to the UR5 base frame by the "pclTransform" package, which is adapted from the "ROS-point-cloud-Transformer" package by github user "petpetpeter" [5]. This package transforms the frame by utilizing the pcl_ros transform package, a bridge for point cloud processing in ROS.

Once the frame is properly transformed, it is then converted into a point cloud file by the "pcdCreate" package. This package not only reads in the transformed frames through ROS and saves them as point cloud files, but it also performs down sampling and filtering with the voxel grid filter from Point Cloud Library. This filter works by creating a 3D voxel grid over the input point cloud data. Then, in each voxel, all the points are approximated by their centroid. This not only significantly reduces the amount of data by an order of magnitude depending on the selected voxel size, but it also more accurately represents the surface of the underlying object compared to quicker voxel center filters. After applying the voxel grid filter, a radius filter of 1 meter is applied to the point cloud. The rationale behind this filter is that the UR5 has an arm length of 0.85 meters. As such, the UR5 cannot collide with any static obstacles outside of this length from the base.

There are several limitations to the way the point cloud is acquired. The first limitation is that the quality of the resulting scan is highly dependent on the quality and number of scan positions selected by the end user. For example, taking too few scans will result in not capturing all the obstacles near the UR5. Likewise, a lack of variety in scan poses will also result in not fully capturing the full shape of obstacles near the UR5. Thus, the end user must take great care in making sure the scan positions will cover the full operating scene. Another limitation to this approach is that the speed and fidelity of the point cloud scan is dependent on the order and parameter selection of the voxel grid and radius filter. A smaller leaf size for the voxel grid filter will result in higher fidelity of object surfaces but also incur a significant computational penalty during object extraction due to more remaining points.

3.3. Object Extraction

After the point cloud scan is acquired, the point cloud is then partitioned into objects using a multi-level clustering method. The purpose of this method is to bridge the gap between raw point cloud data and a collision object representation that can be used by CollisionIK to avoid obstacles. Thus, it was important to implement a method that used shapes natively implemented in CollisionIK already. CollisionIK supports cuboids, spheres, and meshes, and thus any object extraction method needed to convert the point cloud into these shapes.

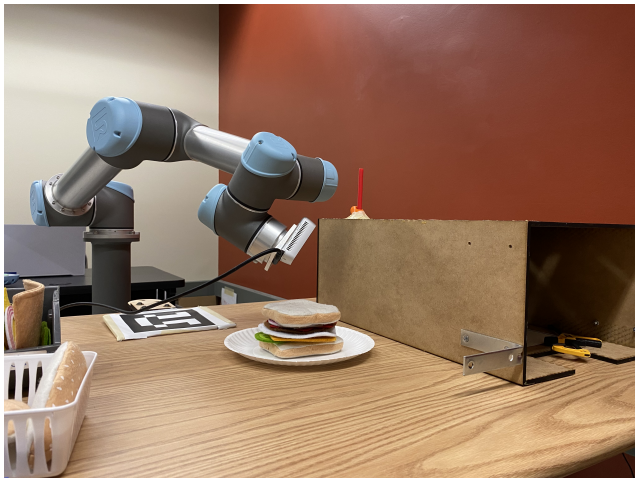


Figure 2. The UR5 scanning the working environment with the D435 camera.

I ultimately selected cuboids as the shape to use for a few reasons. The first is that compared to spheres, cuboids have a tighter fit to the base object it is bounding [6]. Furthermore, cuboids are much quicker to extract from portions of the point cloud compared to meshes. This was an important consideration as it is a goal of this method to eventually be extended to dynamic obstacles as well. Thus, the multi-level clustering method utilizes cuboids to represent the different obstacles in the environment.

Another consideration made was between axis-aligned bounding boxes (AABBs) or oriented bounding boxes (OBBs). The key difference between these two-cuboid based bounding boxes is that AABBs are aligned with the global axis (the UR5 base frame) whereas OBBs are aligned with the primary axis of the object it bounds. While OBBs have the advantage of being rotationally invariant, AABBs are quicker to draw and perform collision tests on. When extending to dynamic obstacles though, OBBs are most likely the best choice since their rotational invariance means they are very quick to update each frame. Therefore, the multi-level clustering method seeks to partition the environment into a collection of AABBs because they produce a tight fit, are quick to draw, and are natively supported by CollisionIK.

However, partitioning the environment into a collection

of AABBs is tricky and can be computationally expensive. The first method I tried was to utilize K-means clustering. This approach essentially took the entire point cloud as an input, separated it out into k clusters, and then drew an AABB around each cluster by taking the minimum and maximum point along each axis.

This approach had several key limitations, however. K-means clustering has a time complexity of $O(n^2)$, and as such, naively inputting the entire point cloud led to significantly long runtimes. Also, since K-means has a natural tendency to create spherical clusters, it will often cluster portions of two separate objects together if they have an irregular shape. The last major limitation to this approach was that if k was not sufficiently large, there was a lack of surface accuracy in the final environment representation.

The next attempt to address these limitations was to introduce the idea of multi-level clustering. The approach works by first clustering the entire point cloud into a few large clusters. Then, each of these large clusters are then clustered into smaller clusters. This has the inherent advantage of significantly improving runtime as the input size is significantly reduced. However, this approach did not resolve the issue clustering different objects together, and it still had poor surface accuracy.

The final approach used was to transition from K-means to Mini-batch K-means clustering and to introduce Density-based spatial clustering of applications with noise (DBSCAN). Mini-batch K-means can be advantageous to K-means as with each iteration, it only takes a mini batch of the input data to update clusters [4]. This was empirically shown to significantly improve runtime on the point cloud data used for this project. Meanwhile, DBSCAN is a clustering method that groups points closely packed together into a cluster. DBSCAN is essentially quite good at finding arbitrarily shaped clusters, something K-means struggles with. DBSCAN is also robust to outliers and does not require one to specify the number of clusters beforehand.

The approach is then to first input the entire point cloud into the DBSCAN method. This will quickly and robustly partition the environment into several large arbitrarily shaped objects that are separable, such as a table near a wall. An example of the DBSCAN results can be seen in Figure 3, where DBSCAN seperates each of the disconnected surfaces automatically. Then, the clusters found by DBSCAN are input into Mini-batch K-means to decompose the arbitrarily shaped clusters into AABBs that can then be input into CollisionIK. This method has the advantage of being significantly faster as both Mini-batch and DBSCAN have quite short runtimes, on the order of about half a second. Furthermore, DBSCAN addresses the issue of separating out distinct objects with its notion of reachability.

However, there are still a few limitations unsolved by this multi-clustering method. The major limitation is that

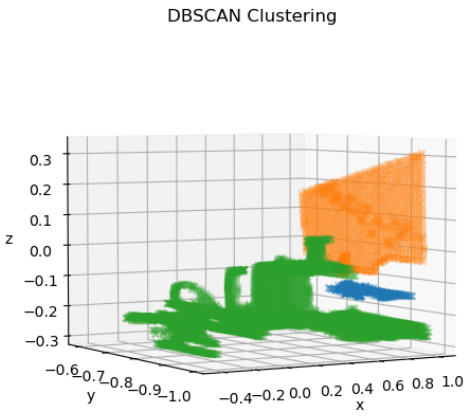


Figure 3. DBSCAN applied to the point cloud.

for this method to run near real-time, at about 15Hz, the amount of AABBs drawn must be significantly reduced. This in turn degrades the quality of surface accuracy, which is detrimental to use cases that requires careful precision along an object surface. It thus is in its current state unable to handle dynamic objects without trading accuracy.

Another limitation to this approach is that compared to some occupancy grid methods such as OctoMap, it does not take advantage of ray-casting methods. This means that the clustering methods makes no attempt to label unobserved spaces as occupied or not. This can be detrimental when using CollisionIK as areas that might be occupied but blocked from view by other objects may be labeled as unoccupied. As a result, CollsionIK may try to move the robot into these regions and collide with these unobserved areas. Thus, this clustering method must run faster to be used in dynamic obstacle avoidance, and it should leverage ray-casting methods to be more accurate and avoid inadvertent collisions.

3.3.1 Clustering Design Analysis

This section will compare the clustering algorithms and hyper-parameters selected for the final system design.

One consideration made was in regards to the ”p-value”, or the percentage of outliers excluded when drawing the bounding box. For example, a p-value of 10 signifies that only points in the 10th to 90th percentile are included in creating a point cloud. This is to account for both the clustering algorithms including points that are not quite close to the cluster centroid as well as noise from the point cloud itself. Figure 4 shows an example of the bounding boxes create for p-values of 1, 4, and 10.

An important comparison to analyse was whether DBSCAN used with Mini-Batch K-Means offered any advan-

tages to classic K-Means. For this comparison, the point cloud shown in Figure 5 was used. Figure 6 shows the K-mean clusters produced by selecting k values of 24, 48, and 99 respectively. Figure 7 shows the respective bounding boxes produced around these clusters as well. Figure 9 shows the clusters produced by applying Mini-batch K-means clustering to each of the clusters produced by DBSCAN for k values of 8, 16, and 33. Because DBSCAN produces 3 clusters in this example, this results in 24, 48, and 99 clusters for a fair comparison to K-means clustering. Figure 9 shows the respective bounding boxes produced around each cluster. Table 1 compares both the total volume encapsulated by the bounding boxes for each method as well as the total runtime. From the results, K-means clustering is more efficient at producing tighter bounding volumes for the same number of clusters. However, Mini-batch and DBSCAN is much faster, with a speedup of 4.5x when creating 96 clusters of the environment. Thus, it is dependent on the target use case whether to use classical K-means or Mini-batch with DBSCAN. Further analysis is required for whether different combinations of methods produce more optimal results for either bounding volume efficiency or runtime.

Clustering Performance Comparison		
Method	Volume (cm^3)	Runtime (s)
K-means, k=24	**563**	0.93
K-means, k=48	**380**	1.44
K-means, k=96	**236**	2.80
Mini, k=8 + DBSCAN	762	**0.37**
Mini, k=16 + DBSCAN	610	**0.56**
Mini, k=33 + DBSCAN	379	**0.61**

Table 1. Clustering performance by algorithm based on total volume enclosed and total runtime. In general, classic K-means produces a more tight fit for the same number of cluster. However, Mini-batch and DBSCAN are significantly faster.

3.4. Collision Avoidance

Once the object bounding boxes are written to CollisionIK, the UR5 is ready for real-time teleoperation with environmental collision avoidance. To run CollisionIK, the user must first launch the solver. Then the user can launch the keyboard controller and the mover package for the UR5 to start responding to teleoperation commands. The UR5 should now be able to move to desired positions in real-time while avoiding any static obstacles observed from the point cloud scan and object extractions steps. Figure 10 shows the collision objects represented by CollisionIK and their positions relative to the UR5.

In terms of the project, however, CollisionIK had several limitations that made it difficult to implement environmental collision avoidance. One limitation is that CollisionIK

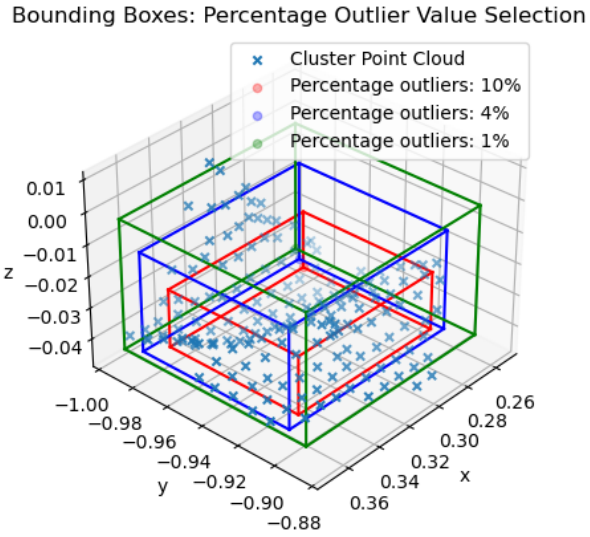


Figure 4. Percentage outlier hyperparameter comparison for bounding boxes for values of 1, 4, and 10 percent.

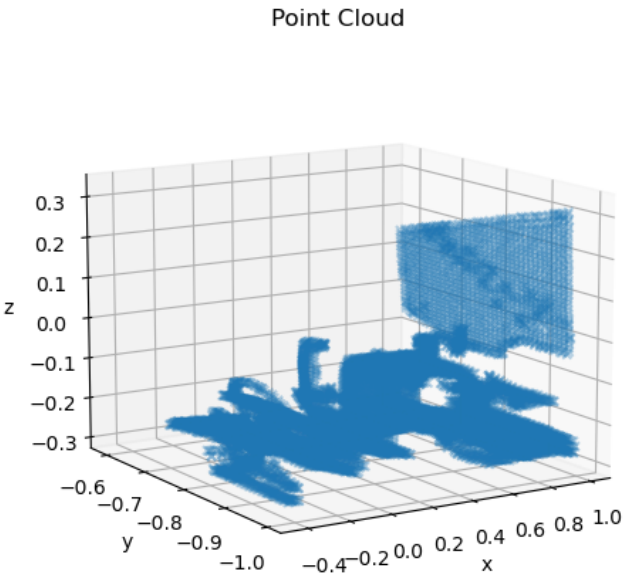
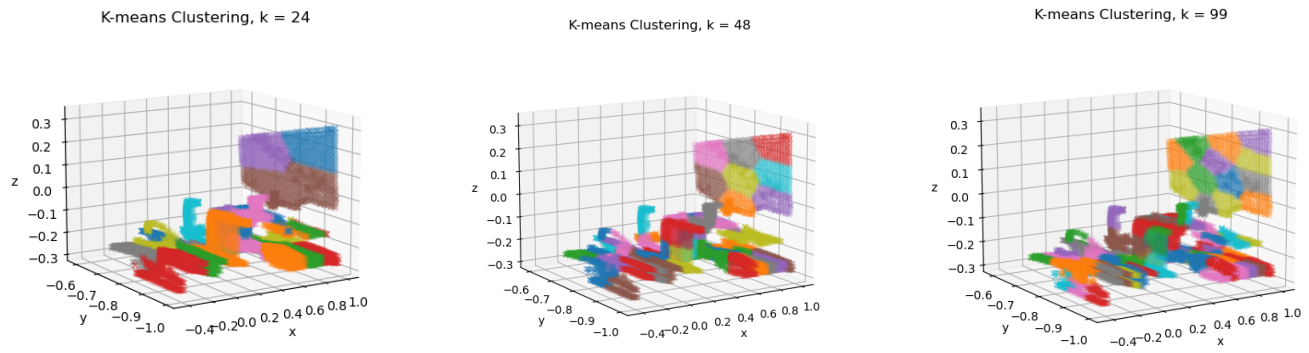


Figure 5. The complete point cloud before clustering extraction.

does not support many types of collision objects and does not support any occupancy gridding. As a result, it can be difficult to implement CollisionIK on any project that requires it to interface with sensor data. Another limitation of CollisionIK is that it is not clear how many collision objects it can support before it no longer can run at real-time. For example, while it is clear a raw point cloud would be too many collision objects to handle, it is unclear if an occupancy grid is also too much for CollisionIK to handle. Oc-

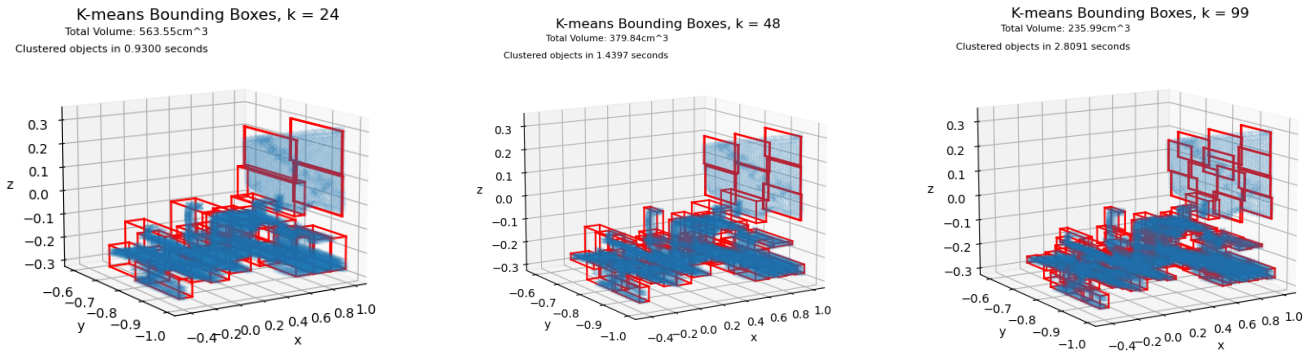


(a) K-means clustering for k=24.

(b) K-means clustering for k=48.

(c) K-means clustering k=99.

Figure 6. Comparison of K-means clustering for k values of 24, 48, and 99.

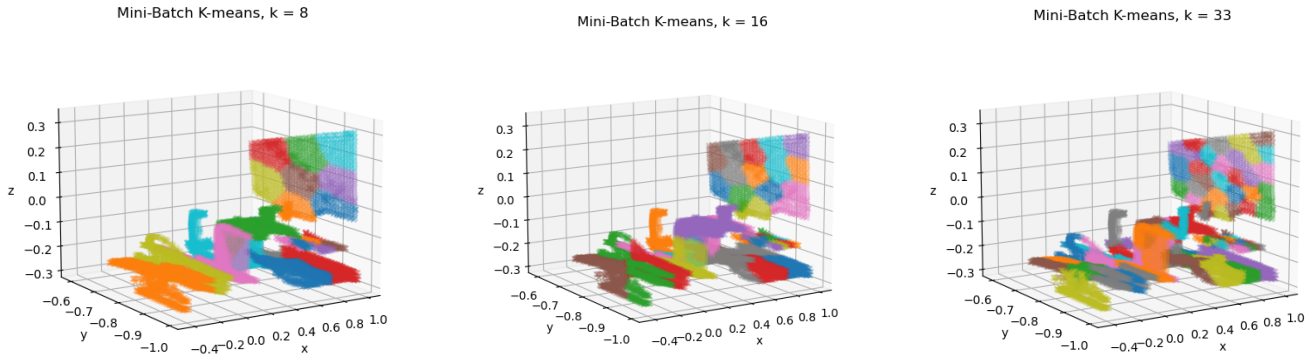


(a) Bounding boxes from K-means clustering for k=24.

(b) Bounding boxes from K-means clustering for k=48.

(c) Bounding boxes from K-means clustering k=99.

Figure 7. Comparison of the bounding boxes from K-means clustering bound for k values of 24, 48, and 99.

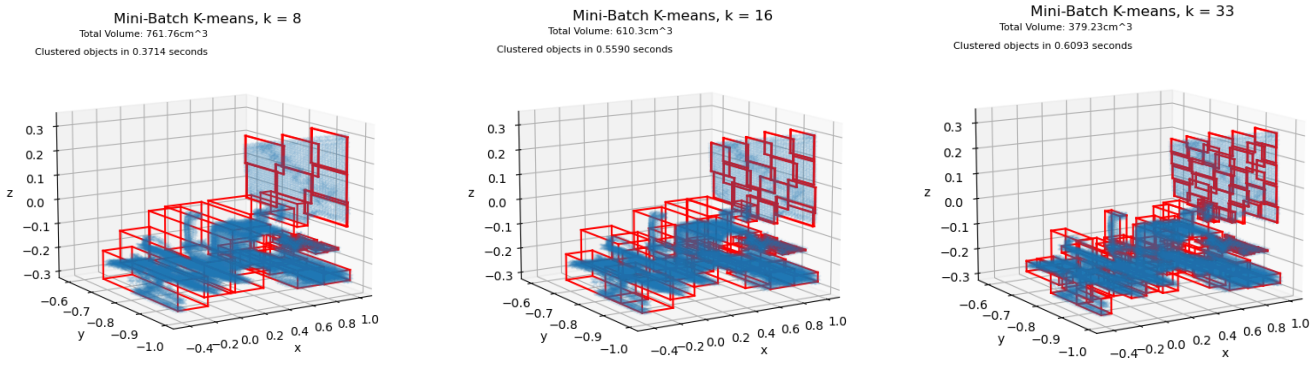


(a) Mini-batch K-means clustering for k=8 and DBSCAN.

(b) Mini-batch K-means clustering for k=16 and DBSCAN.

(c) Mini-batch K-means clustering for k=33 and DBSCAN.

Figure 8. Comparison of Mini-batch K-means clustering for k values of 8, 16, and 33 with DBSCAN.



(a) Bounding boxes from Mini-batch K-means clustering for k=8 and DBSCAN

(b) Bounding boxes from Mini-batch K-means clustering for k=16 and DBSCAN.

(c) Bounding boxes from Mini-batch K-means clustering k=33 and DBSCAN.

Figure 9. Comparison of the bounding boxes from Mini-batch K-means clustering bound for k values of 8, 16, and 33 with DBSCAN.

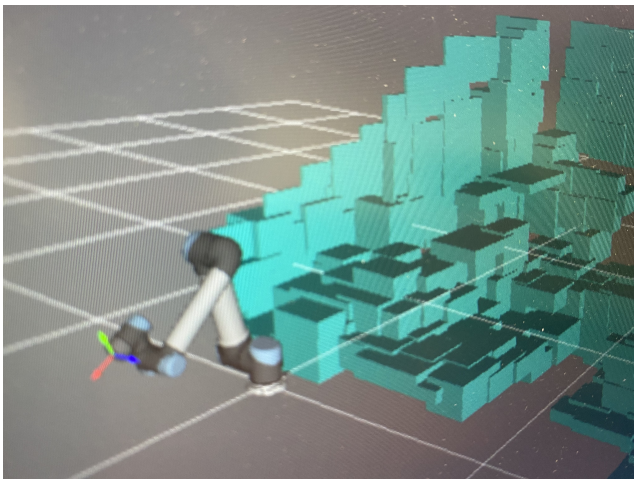


Figure 10. The CollisionIK representation of the extracted collision object environment.

cupancy grids also have tunable hyperparameters that can control how many voxels are created and knowing the performance limitations of CollisionIK would be useful if attempting to bridge CollisionIK with these grids. The last major limitation with CollisionIK is that adding dynamic obstacles to its environment representation is quite tricky. At the current moment, the best way to add dynamic obstacles is to predefine obstacles and place them far away from the UR5. When a new dynamic obstacle comes into view, this predefined dummy obstacle can then be redefined to bound the new dynamic obstacle. However, it is unclear if this approach is resilient to a variety of situations such as an obstacle changing size or rotation. These limitations may be important to address when seeking to use CollisionIK for dynamic obstacle collision avoidance.

4. Discussion

In this work, a system that can avoid environmental static objects during teleoperation of the UR5 was presented. This section discussed the implications and limitations of the system as well as potential avenues for expanding it.

4.1. Implications

This system scans the surrounding environment and process the raw point cloud data for object extraction. The system can then extract objects from the point cloud at varying levels of surface fidelity using a multi-level clustering approach. The system adds these objects to CollisionIK's environment representation so that the UR5 can avoid the static environmental objects during teleoperation.

One of the main strengths of this system is that it can generalize to a variety of environments and does not require any prior assumptions. It also succeeds in being tunable for different required surface fidelities based on the required use case. For example, if surface fidelity is important, the end user can tune the clustering hyperparameters and alter the different clustering methods used in each level to improve the object extraction.

This system could benefit different areas of robotics. Teleoperation use cases such as domestic support, cobot-based manufacturing, and telehealth could all utilize this system to detect and avoid obstacles during operation. The clustering-based object extraction method can also be used in other fields such as unmanned aerial vehicle control to quickly determine unoccupied regions for flight. This system helps bridge the gap between control methods and environment construction, and is useful for any scenario that does not have dynamic obstacles.

4.2. Limitations

A key weakness of this method is that the clustering methods do not produce high levels of surface fidelity compared to occupancy grids. The AABBs generated by the clusters introduce quantization which do not accurately represent the continuous nature of surfaces.

Another key weakness of this method is that the clustering is too slow to be extended to dynamic obstacle avoidance. With the fastest speeds ranging from 2-10Hz, the clustering method is unable to recognize and process obstacles fast enough for the UR5 to safely avoid. Thus, while this system can perform static collision avoidance and is generalizable to a variety of environments, it lacks surface fidelity for more precise use cases and the speed to be extended to dynamic collision avoidance.

4.3. Future Work

There are several directions someone could take to improve bridging the gap between environment mapping, robotic teleoperation, and collision avoidance, including for dynamic obstacles.

4.3.1 Hyperparameter Exploration

One key weakness of this work is the lack of quanitative analysis on the performance of the system and the various tradeoffs present based on the hyper-parameter selection. For example, it would be interesting to explore how the selection of k or the choice of DBSCAN parameters impact the quality of clusters in various environmental scenarios. Further quantitative investigation should be made into how the hyper-parameters impact the speed, accuracy, and memory consumption of the system to help guide future design decisions.

4.3.2 Mesh Filtering

Another area of potential improvement for the system is improving the per frame object extraction runtime, as the current speed of 0.1 to 0.5 seconds is too slow. One idea is to first perform a high-fidelity yet slow static object extraction initialization step. A mesh filter could be created from the static objects so that the point cloud frames during runtime would ignore any static regions. By filtering out these regions, the size of the point cloud per frame would be considerably smaller and only include any dynamic obstacles. This could potentially improve the runtime of point cloud processing and object extraction per frame to a speed suitable for dynamic collision avoidance for real-time teleoperation.

4.3.3 Object-oriented Bounding Boxes

Another way to also extend the current work is to switch from using AABB's to OBB's. OBB's have the advantage of having both translational and rotational invariance, which means they do not need to be recomputed. CAB demonstrated how to use an OBB tree to produce a two times speedup in creating a bounding volume hierarchy of a human arm moving compared to an AABB approach. Recreating a technique like CAB could thus be very useful in teleoperation where a human might be working near the UR5.

4.3.4 Occupancy Grids

The other approach to extending this work into dynamic obstacles could be to abandon the clustering strategy and to extend CollisionIK to work with occupancy grids. The advantages of using an occupancy grid are that it has high surface fidelity and includes the notion of free, unoccupied, and unexplored space. I originally had abandoned using OctoMap because the runtime speeds were not sufficient for real-time teleoperation. However, RCIK successfully demonstrated that it is possible to use an occupancy grid for collision avoidance and achieve real-time speeds for dynamic obstacles. Therefore, it could be possible to adapt the occupancy grid method used by RCIK to work with CollisionIK.

One of the challenges that would need to be addressed, however, is how to represent an occupancy grid in CollisionIK. The natural approach would be to be represent each voxel as a cuboid in CollisionIK. One would need to study if CollisionIK can handle that many cuboids during runtime, or if hyperparameter tuning of the voxel size and range of collision checking would enable real-time speeds.

Another challenge that would need to be addressed is working around or extending CollisionIK's dynamic obstacle interface. As previously mentioned, it can be quite tricky to add dynamic obstacles or update them between frames if they vary by size or rotation. However, both challenges seem tractable and thus using an occupancy grid is most likely the best approach to bridge the gap between environmental mapping and dynamic collision avoidance during robotic teleoperation.

5. Reflection

It is important to reflect on how a project went and what worked in order to improve for next time. In this section, I discuss how I approached and completed the project. I also discuss the various technical and non-technical skills and insights I gained in the process of completing this project. I close this section by evaluating the project itself and my performance, and the ways I could improve for next time.

5.1. Work Done

This project took an extraordinary amount of effort and experimenting to get working. I review in this section the approach I took to complete the project after switching work from the Kinova MOVO to the UR5.

The first step for this project was to perform a preliminary literature review of collision avoidance. I read different papers on how other teams were approaching the problems of object detection, collision avoidance, and motion planning. It was from this literature review I found motion solvers such as CollisionIK and RCIK as well as environment recognition methods such as OctoMap and DBSCAN.

The next step I took was to perform preliminary experiments in environment mapping methods. I first tested OctoMap, and I found that OctoMap does an excellent job recreating the environment. However, it could only run at 5 Hz. Furthermore, it was unclear how to convert the voxels into a cuboid representation supported by CollisionIK. I also tried experimenting with the Point Cloud Library implementation of DBSCAN. I found it could identify objects, but it did a poor job of recreating the environment itself, namely planar surfaces such as tables and walls. I also tried inputting a raw point cloud into CollisionIK, but that ran far too slow, even with a point cloud of only approximately 500 points.

I then switched my attention to upgrading my computer. In testing the environment recognition methods, I found my current PC was far too weak to effectively develop and test for this project. This was quite challenging, as the new PC I bought did not have a Wi-Fi card supported by Ubuntu 18. This led to me using a virtual machine on my new PC and debugging networking issues with the virtual machine.

Once the virtual machine was set up, I focused on setting up the calibration method Seth had found. I then proceeded to work on capturing a point cloud from the D435 camera. The most challenging aspect of this step was that the camera would disconnect often from my PC and had trouble interfacing with my virtual machine, often taking up to ten attempts to even be usable. Once I was able to capture the point cloud and convert it into a pcd file, I also added a voxel grid filter to down sample the data.

One of the major steps for this project was implementing a method to convert the point cloud data into an environment representation. From the earlier experiments, OctoMap, DBSCAN, and raw point clouds all had serious limitations. Upon Danny's recommendation, however, I decided to use a clustering-based environment recognition method. The original idea was that because CollisionIK chiefly supports cuboids and spheres as collision objects, I could cluster the point cloud data and bound each cluster as a cuboid or sphere. Then, each of these bounding volumes can quickly and easily be written to CollisionIK. The initial method I wrote simply used scikit-learn's implementation of K-means, and I tested it out on a desk scene from the Point Cloud Library. I also tried out different values of k and investigated using k-selection methods such as the elbow method. However, the issues with choosing low values of k suggested by the elbow method is that the resulting bounding boxes have a very loose fit. As such, I decided to use a k of roughly 100 and then created AABBs for each of these clusters to produce a tight fit. My initial method then wrote these AABBs to CollisionIK's settings file automatically so that they'd be initialized automatically when running the solver.

Now that I had a method to convert point cloud data into an environment representation, I focused on aligning the point cloud data to the UR5 base frame. Originally, I acquired the transformation by calibrating the D435 camera on a tripod to the UR5 base frame using the eye-on-base method. I would then manually apply this transformation matrix to the point cloud itself before running the environment extraction method, but I quickly found that the point cloud seemed to be incorrectly by a factor of 90 degrees in two axes and had some translational errors as well.

I first tried rotating the point cloud by a factor ninety degrees in different axes in different combinations, but this was unsuccessful in resolving the rotational errors. I also tried recalibrating a few times to see if this would help the translational errors, but this was unsuccessful.

The next step I tried was to use an already existing project to apply the transformation. I found a github project titled "ROS-point-cloud-Transformer" by user "petpetpeter", and I adapted this into my own project to convert the point cloud data. However, this also did not resolve the rotational or translational errors I was having.

It was only when Pragathi mentioned that the rotational errors were with CollisionIK. I had been visualizing the rotated point cloud with the CollisionIK visualizer, but the issue is that CollisionIK has an incorrect frame tree representation. Thus, to resolve the frame tree issue, I created a branch of the frame tree that connected the real UR5 base frame to the "virtual" base frame represented in CollisionIK. I then applied the transformation so that the point cloud would be transformed first from the camera frame to the real base, and then from the real base to the virtual CollisionIK base. This fixed my rotational issues, but I still had translational issues.

Since I could not figure out why I was still having translational errors, I decided to switch the camera mounting from a tripod onto the UR5 end-effector. The rationale for this by attaching the D435 camera to the UR5 directly, there would be no need to worry about the translation between the two. I thus designed a simple D435 mount, and my friend machined it for me out of aluminum. This resolved the translational error, and I could now go from calibration to extracting collision objects. I also wrote code to move the

UR5 to different scanning positions, so that I could produce a full environment scan. I think overall, resolving the rotational and translational issues was the most difficult aspect of this project, and it was a huge relief when it was finally resolved.

The last step to making a minimum viable product of the system was to figure out how to move the UR5 with CollisionIK. I first wrote code that interfaced with the keyboard controller output from CollisionIK and then sent commands directly to the UR5 using the servoj command. This took some experimentation to find optimal control parameters to move the UR5 smoothly. One issue that arose, however, was that the CollisionIK representation of the UR5 movement was misaligned with the real UR5 movement. Pragathi gave me the offsets required to correct for the joint angle errors CollisionIK had, and applying the offsets corrected this error. Thus, the UR5 could now move with CollisionIK, as seen in Figure 11, and a minimum viable product that could recognize and avoid static obstacles was complete.

Two of the main issues I wanted to fix after finishing the minimum viable product was the speed and accuracy of the collision object extraction method. Upon initial tests, the system would take about 60 seconds to extract collision objects and the UR5 would still hit objects during teleoperation.

In order to resolve the accuracy of the method, I tuned the hyperparameter responsible for ignoring outliers of a cluster when creating the AABBs. I settled on using any value between the 1st and 99th percentile for creating the AABB, as can be seen in Figure 4. While this may include a few outliers, it generally recreated objects accurately and significantly reduced the occurrence of collisions during teleoperation compared to the original 10th to 90th percentile choice.

Next, in order to resolve the speed of the clustering extraction method, I experimented with using different clustering techniques. One technique I tried was to first create a few clusters and then break down those clusters into more clusters using K-means at both stages. To further speed up this approach, I then transitioned from K-means to Mini-batch K-means, which led to runtimes from 2.5 seconds to 1 second. Lastly, noticing that the first clustering stage is meant to break the scene down into its base components, I switched to using DBSCAN as its density-based clustering approach is excellent in identifying the base components. This switch improved the runtime from 1 second to anywhere from 0.1 to 0.5 seconds based on the surface fidelity required from the second stage mini-batch k-means. This thus concluded my work on the project, as it reached my expectations for speed and accuracy and seemed like a good point to conclude the exploration of clustering-based methods for collision avoidance.

5.2. Learning

Working on such a large and comprehensive project gave me a multitude of opportunities to learn and develop both technical and non-technical skills.

5.2.1 Technical

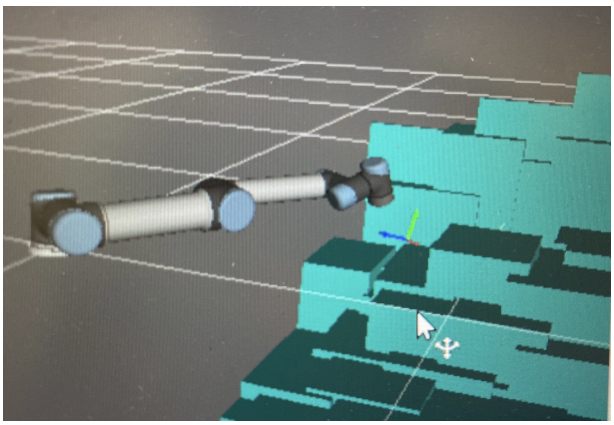
I learned a great deal of technical knowledge with this project given the wide range of technologies required for everything to work together. The first major subdiscipline I learned was regarding calibration and transformation. I learned about eye-on-hand and eye-on-base calibration. I also learned about how to apply transformations as well as how to utilize ROS tf trees to keep track of reference frames, and how difficult it can be to debug transform issues when a tf tree is incorrectly configured as is the case with CollisionIK.

I also learned a lot about point clouds. It was surprising to see the lack of methods that directly handle depth information compared to the multitude of methods available for RGB image processing. I got experience with processing, downsampling, and filtering point clouds as well. I also got exposed to the Iterative closest point algorithm and how it could be used for point cloud registration, especially when creating a point cloud scan.

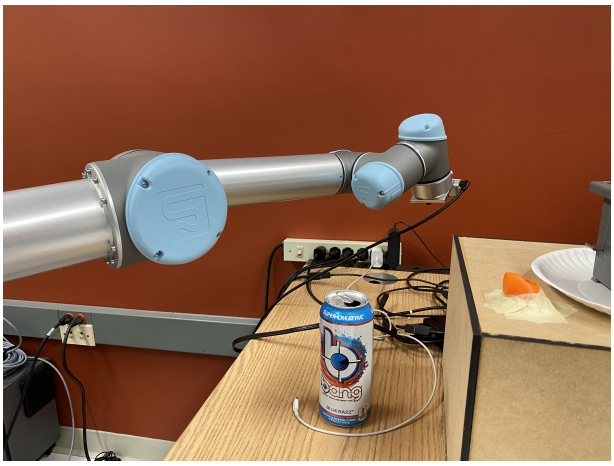
One of the concepts I was most surprised to have learned so much about was bounding volume hierarchies. I had originally assumed using a bounding box or sphere was sufficient for the purposes of the project. However, in reading about AABB's, OBB's, bounding box trees, spheres, ellipses, and meshes, I discovered that the choice of bounding volume hierarchy can present significant advantages and disadvantages in the design of the underlying collision avoidance system.

I also read a great deal of literature regarding occupancy grids. It was interesting to see how kd-trees can be extended to represent the environment in a voxel-like manner, and some of the considerations the authors would make to optimize speed, accuracy, and memory consumption.

One of the technologies I had the most exposure with during this project was point cloud clustering techniques. In the process of developing and optimizing my collision avoidance system, I had to explore the advantages and disadvantages of each clustering technique. I spent time experimenting with different hyperparameters, as well as trying out different orders of clustering techniques when creating the multi-level method. It was also interesting discovering that the conventional techniques for clustering parameter selection was highly dependent on the use case including dimensionality and density, and as such, not every piece of advice worked well. For example, the elbow method for k selection for K-means worked quite poorly in creating a tight representation of the environment.



(a) CollisionIK representation of the UR5 avoiding obstacles on a table.



(b) UR5 avoiding obstacles on a table using CollisionIK

Figure 11. Side-by-side comparison of the UR5 in CollisionIK and in real-life.

I also got to deep dive into CollisionIK when working on incorporating collision objects extracted from point cloud data. I also read into other approaches for static and dynamic collision avoidance such as RCIK and saw some other methods that were applicable to drone control. I think overall, it was extremely fascinating to read and explore how different data structures and techniques from other aspects of computer science and engineering could be combined to solve this complex problem.

5.2.2 Non-technical

This project also took an extraordinary amount of time, being one of the most comprehensive and longest projects I have ever done, and as such it taught me many skills beyond the technical. I think one aspect that was interesting was how many academic works solve such complex problems, but do not spend much time considering the use case or how they would fit in with other works. I often found myself asking the question of what the purpose of laboring to create amazing academic works is if no one can easily use them or apply them to their own works. It is most likely quite evident with the literature review I performed for this project as collision avoidance in real-time is a difficult problem, and it is more tractable to focus on one aspect of the problem. I think I really got to learn and appreciate how academia works, and how the literature expands over time as we explore and understand different solutions.

I also learned, in my own struggles to figure out the angle my project sought to tackle, how to be a better researcher. The experience of completing a project helped me get more experience with researching, expanding, and testing my ideas. It also helped me pose research questions and how to then tackle the process of attempting to answer all of them. I found it helpful to write a list of questions I currently had regarding what I was working on, such as when I was wondering why my collision object extraction method was running quite slow. It helped me break the problem down into smaller components and understand where exactly the method may be failing and what I could try to resolve it. I often found myself, especially early on, overwhelmed by the scale of this project and feeling like I had no clear direction on how I could solve it. By breaking the problem down into key questions, I was able to create smaller tasks and therefore reduce the amount of stress I was feeling. Stress was the biggest roadblock I had during this project and learning techniques to manage it in the context of the research and managing a project was an invaluable skill to have developed.

I also learned where I would like to go after having completed this project. I first would like to work in an environment where collaboration is part of the work culture. I remember how helpful the moments were when other people in the lab pointed me in the right direction. This project's direction, for example, owes a lot to Danny's advice in which directions to pursue and what I could reasonably expect to accomplish. I sincerely hope that the job I start at the end of the summer will value collaboration, as I believe it is essential to developing as an engineer. I also learned that doing a PhD is probably not the right path for me. For most of graduate school, I have been on the fence whether to eventually do a PhD after a few years working in industry. The experience of conducting this project, however, has shown me that I would probably not be successful or happy in such a program. I had a terrible time in doing a roughly 6-month long project. The prospect of doing a 4-year project, especially where it is expected that the final work is novel and pushes the field forward, seems both daunting and incredibly stressful. While I think research can be incredibly rewarding as

one seeks to uncover the answers to whatever they are curious about, the type of work and energy that goes into it is simply not for me. I am grateful that this project gave me the experience to try out being a researcher for a few months and give me a better idea of where I would like my career to go and how I would like it to look.

5.3. Self-evaluation

Lastly, with this project being as late as it is, I think it is crucial to evaluate my own performance. This is essential to learning from my successes and mistakes and continue my professional development as an engineer. Overall, I think I did alright with the work I accomplished and the results I saw with the system. I had a great deal of unlucky circumstances occur during this project, from my laptop breaking repeatedly, having to take on an extra job to financially support myself, and being hospitalized briefly. However, there are several key areas I could improve on with the next large project I work on.

The first key aspect I could improve on is my project organizational skills. I feel that I did not approach this project in a very focused manner. I should have started by spending more time reviewing the literature, consulting with my peers, and coming up with a proposal on how to tackle the project. I also think I should have devoted more time in developing quantifiable performance metrics to compare the different methods I was experimenting with. While visualizations are helpful, I did not have a good way of analyzing the trade off of speed versus the quality of the environment reconstruction.

I also should have set more clear goals and aspirations for the project, as this would have helped me focus on the components most important for the system to work. Instead, I got quite overwhelmed by how daunting the project seemed and opened too many avenues to try and work on in the hopes one might work. I also did waste some time debugging errors inefficiently. For example, when dealing with the calibration issues, it was a waste of time to attempt to correct the errors manually instead of trying to understand what was really happening at the system level.

I also think my performance in writing the project report itself to be a bit poor. While much of the delays in writing the report can be attributed to burnout and some external unlucky circumstances, I also did not have a clear strategy on how to write the report or how I wanted it to be structured.

I think the last area of improvement is being disciplined in my work. I found that it was easy to get a great deal of progress made when I felt inherently motivated in the aspect of the project I was working on. However, when the motivation was not present, especially as I realized this project did not succeed as much as I would have hoped, I had trouble getting any work done. I would have periods of a few days in a row of not being able to accomplish anything meaningful or writing anything substantial. In future projects, I would like to work on being more disciplined and consistent in working and continually making progress irrespective of how motivated I may feel. Every project will have aspects of it that I may or may not enjoy, and it is something I need to get used to.

There are a few key pieces of advice I would give anyone working on something similar to this project. One piece of advice would be to get guidance on CollisionIK and calibraiton. The frame tree and dynamic object interface issues with CollisionIK can be confusing and major roadblocks, and it would save someone a great deal of time to have guidance how to navigate around these problems. The next piece of advice I would give is to stay organized. I would advise them to have clear goals on what they are trying to accomplish as well as a general roadmap on how they would get there. It may take more initial time to plan and research, but it is well worth it in the context of the entire project. Next, I would advise someone to keep track of the work they have done and why one method worked better than another. It helps not repeat trying methods that did not work, not throwing out methods that may be useful in the future, and to understand at the end of the project everything they did to complete the project.

Despite the mistakes I made and the extra time I spent working on the project, I am quite happy with the result. It is personally quite rewarding that I made a full environmental recognition and collision avoidance system both with my own methods and other projects such as CollisionIK. It is also great that I managed to figure out how to improve the speed and accuracy of the object extraction method, and I think I did a good job experimenting with different ideas and creating the multi-level clustering technique. Furthermore, another point of success was that despite not having a clear direction or approach handed to me with this project, I slowly read the literature and experimented my way into understanding a way in solving the problem given to me. Thus, despite the struggles I had these past few months and my own mistakes in how I approached this project, I think I overall did alright with coming up and implementing a working solution and exploring potential future extensions to this project.

References

[1] Euclidean Cluster Extraction. https://pcl.readthedocs.io/projects/tutorials/en/master/cluster_extraction.html#cluster-extraction. Accessed: 2022-7-26. 2

[2] K-Means++ Algorithm for High-Dimensional Data Clustering. https://towardsdatascience.com/k-means-algorithm-for-high-dimensional-data-clustering-714c6980daa9. Accessed: 2022-7-26. 2

[3] K-Means Clustering in Python: A Practical Guide. https://realpython.com/k-means-clustering-python/. Accessed: 2022-7-26. 2

[4] sklearn.cluster.MiniBatchKMeans. https://scikit-learn.org/stable/modules/generated/sklearn.cluster.MiniBatchKMeans.html#sklearn.cluster.MiniBatchKMeans. Accessed: 2022-7-26. 2, 4

[5] Piya Chanapol. Rospointcloudtransformer. https://github.com/petpetpeter/ROS-point-cloud-Transformer, November 2021. 3

[6] Simena Dinas and Jose Bañón. A literature review of bounding volumes hierarchy focused on collision detection. 17:49–62, 06 2015. 1, 3

[7] Matias Hermanrud Fjeld. 3d spatial navigation in octrees with reinforcement learning. Master's thesis, University of Oslo, 2018. 2

[8] David Fridovich-Keil, Erik Nelson, and Avideh Zakhor. Atommap: A probabilistic amorphous 3d map representation for robotics and surface reconstruction. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3110–3117, 2017. 1

[9] Armin Hornung, Kai Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Autonomous Robots*, 34, 04 2013. 1

[10] Mincheul Kang, Yoonki Cho, and Sung-Eui Yoon. RCIK: Real-Time Collision-Free Inverse Kinematics Using a Collision-Cost Prediction Network. *IEEE Robotics and Automation Letters*, 7(1):610–617, 2022. 2

[11] Youngsun Kwon, Donghyuk Kim, Inkyu An, and Sung-eui Yoon. Super rays and culling region for real-time updates on grid-based occupancy maps. *IEEE Transactions on Robotics*, 35(2):482–497, 2019. 1

[12] Gao Chen Lei Zhang. ur5_realsense_calibration. https://github.com/portgasray/ur5_realsense_calibration, 2020. 2

[13] Daniel Rakita, Haochen Shi, Bilge Mutlu, and Michael Gleicher. Collisionik: A per-instant pose optimization method for generating robot motions with environment collision avoidance. *CoRR*, abs/2102.13187, 2021. 2

[14] Harald Schmidl, Nolan Walker, and Ming Lin. Cab: Fast update of obb trees for coillision detection between articulated bodies. *Journal of Graphics Tools*, 9, 01 2004. 2

[15] Simon Vanneste, Ben Bellekens, and Maarten Weyn. 3dvfh+: Real-time three-dimensional obstacle avoidance using an octomap. volume 1319, 07 2014. 2